University of Pittsburgh

Algorithms and Data Structures 2
CS 1501

Spring 2022

Sherif Khattab

ksm73@pitt.edu

(Slides are adapted from Dr. Ramirez's and Dr. Farnan's CS1501 slides.)

Announcements

- Upcoming deadlines:
  - Assignment 2 late due date on 3/30
  - Lab 9 due on 4/1
  - Homework 10 due on 4/4
  - Assignment 3 and 4 due on 4/18
    - Used to be one assignment

Previous lecture …

- Dynamic Connectivity Problem
  - Union/Find data structure

CourseMIRROR Reflections (most confusing)

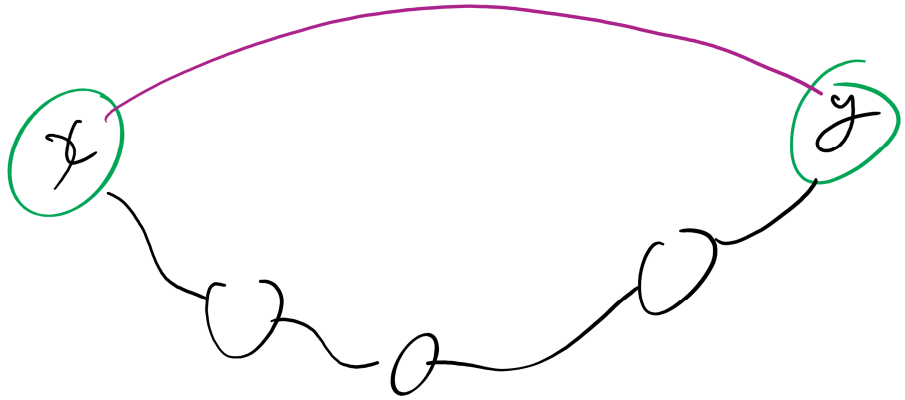
- I still found the heap sort confusing. Going through another example would help
- at the end of making the heap array, why do we switch the first item and the item at the boundary?
- I am still confused why we have to sort the Heap in descending order
- Personally, I feel like the class has gotten very fast lately, it's become harder to keep up with the new material
- The difference between storing UF as an array or as a tree
- using Union/find with Kruskals
- The union part was a bit confusing.
- I was confused on the runtime breakdowns for different prims algorithms
- The most confusing part of class today was applying a priority queue for Lazy Prim's and Eager Prim's algorithms.
- The middle part of lecture. The PQ and lazy/eager implementations were gone over very quickly. Going through an example would help

CourseMIRROR Reflections (most interesting)

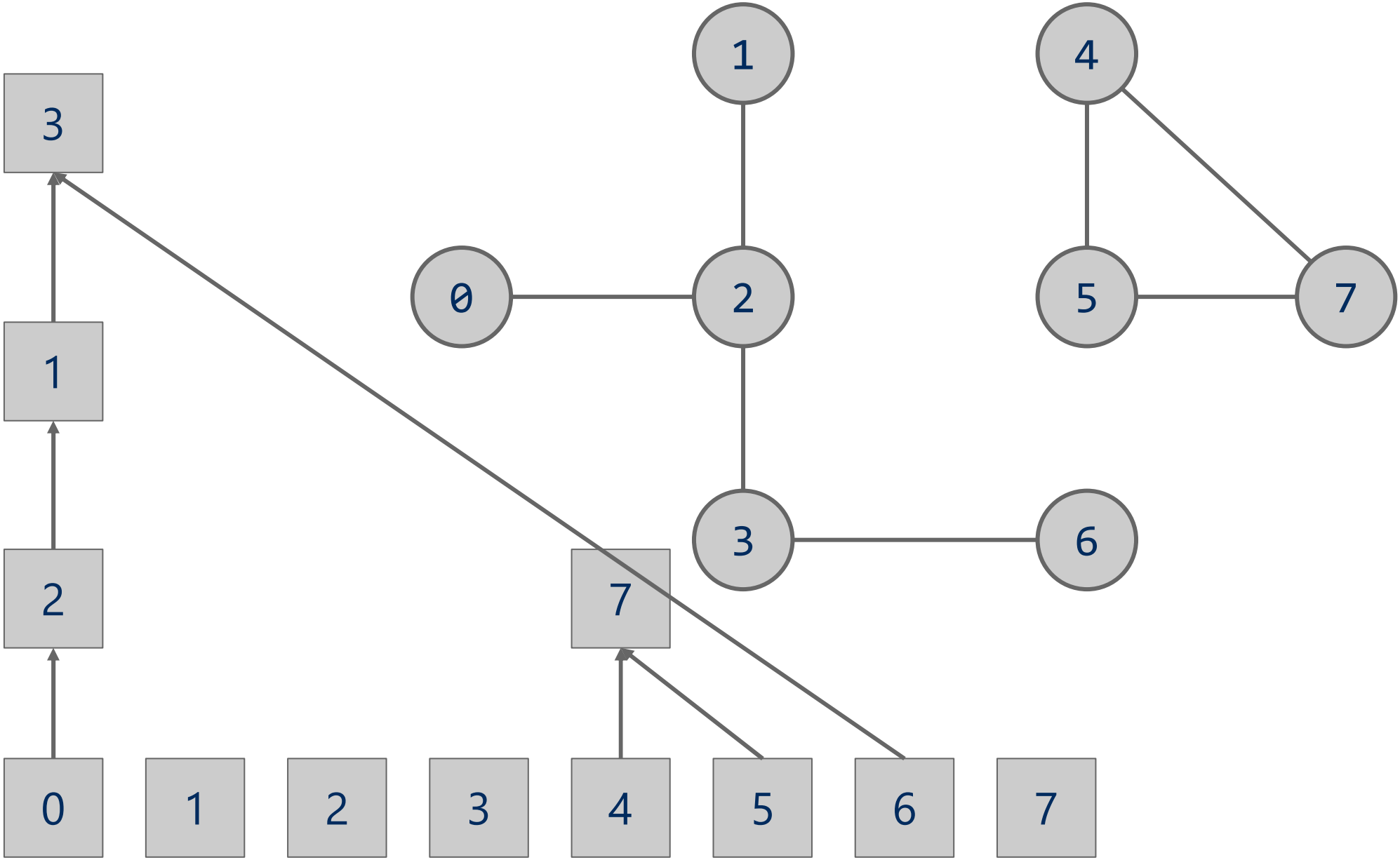
- Applicability of more complicated algorithms to help improve prim's and kruskal's
- I found it interesting how a priority queue could improve Prims Algorithm
- Eager prim for improved runtime
- The runtime efficiency.
- Dynamic connectivity problem
- The connectivity problem and how a graph is a good approach to solving
- Connections of groups and the array of IDs
- The most interesting part of today's class was Union/Find ADT.
- The union examples were fun to walk through
- The new problem of the day, iterative graph building via unions/merges, seems interesting
- code for union find API

Problem of the Day: Dynamic connectivity problem

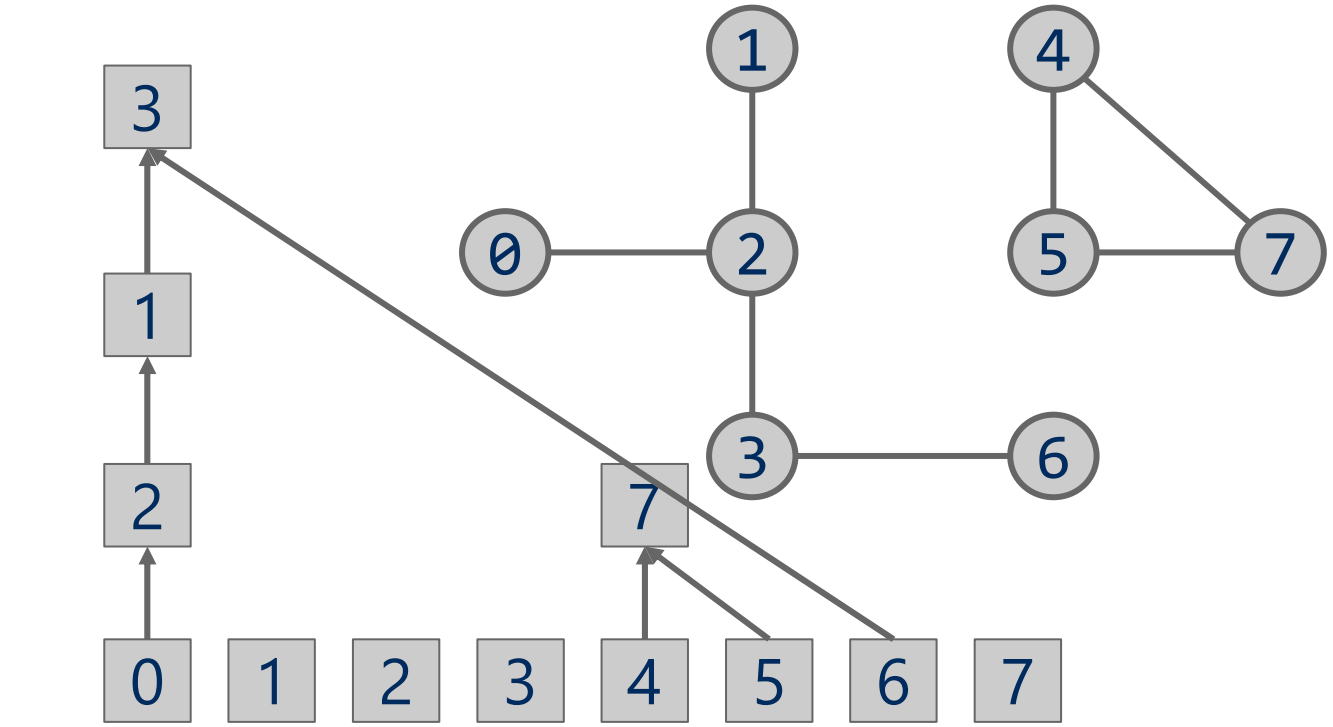
- Input:
  - A set of items initially in separate groups and
  - a sequence of merge/union operations, each operation mering two items
- Output:
  - At any point of time, we can be asked if two items are in the same group
  - Initially, the answer will be NO for any two items because they start in separate groups
- For a given graph G, can we determine whether or
- Can also be viewed as checking subset membershi
- Important for many practical applications

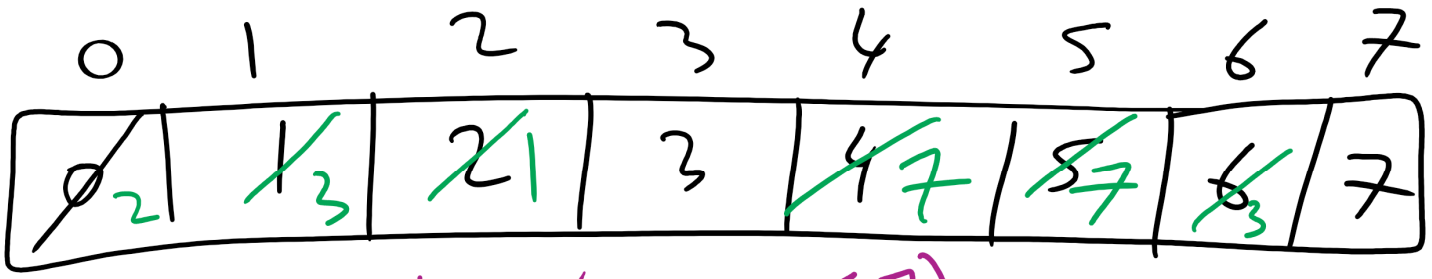


Tree example



Forest of Trees Implementation





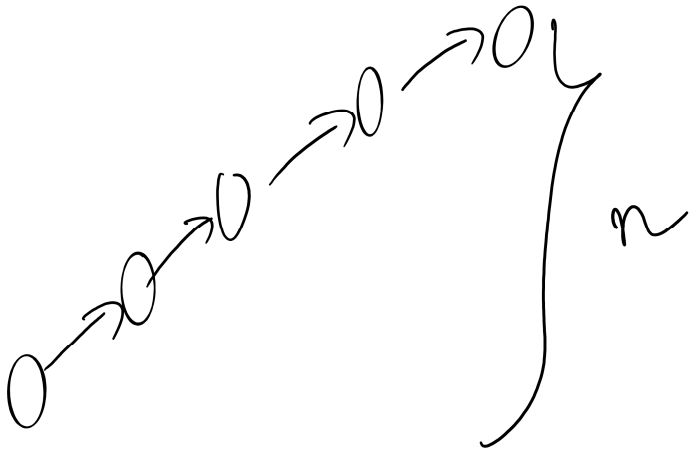

```
while( p != id[p])
    p = id[p];
```

Implementation using the same id array

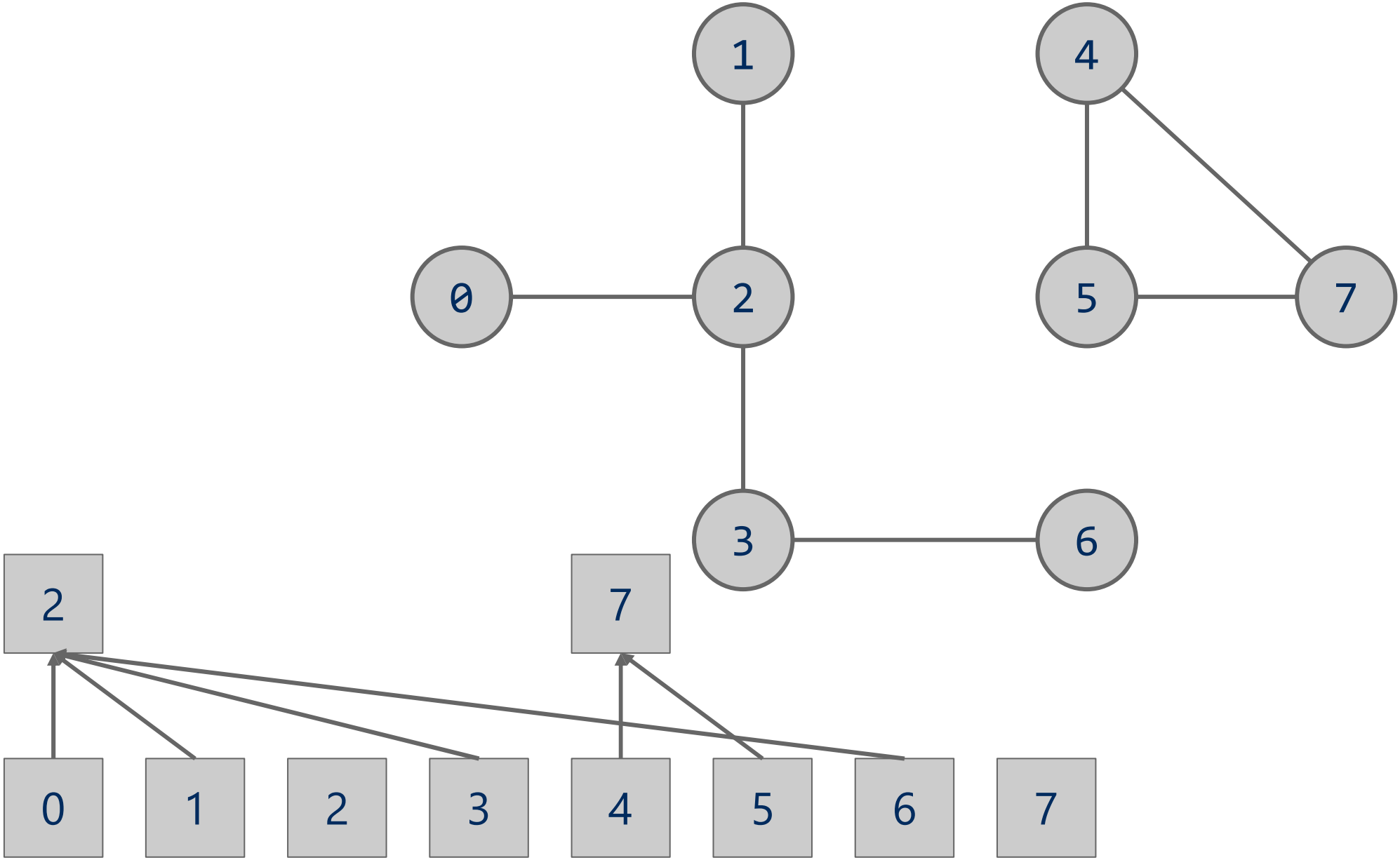
```
public int find(int p) {
    while (p != id[p]) p = id[p];
    return p;
}

public void union(int p, int q) {
       int i = find(p);
       int j = find(q);
       if (i == j) return;
       id[i] = j;
       count--;
}
```

Forest of trees implementation analysis

- Runtime?
  - find():
    - Bound by the height of the tree
  - union():
    - Bound by the height of the tree
- What is the max height of the tree?
  - Can we modify our approach to cap its max height?



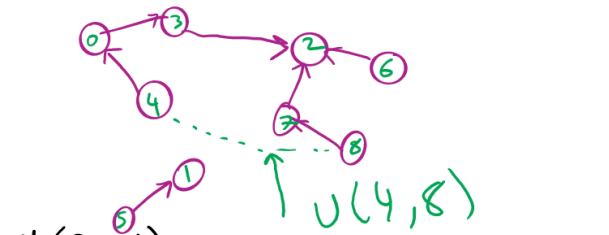
Weighted tree example



Weighted trees

```
public UF(int n) {
    count = n;
    id = new int[n];
    sz = new int[n];
    for (int i = 0; i < n; i++) { id[i] = i; sz[i] = 1; }
}

public void union(int p, int q) {
       int i = find(p), j = find(q);
       if (i == j) return;
       if (sz[i] < sz[j]) { id[i] = j; sz[j] += sz[i]; }
       else               { id[j] = i; sz[i] += sz[j]; }
       count--;
}
```

Weighted Trees Union-Find Example



id (Parent)

3	1	2	3	0	1	2	2	7
0	1	2	3	4	5	6	7	8

find (p)

```
while( p != id[p])
    p = id[p]
return p
```

sz

2	2	4	3	1	1	1	2	1
0	1	2	3	4	5	6	7	8

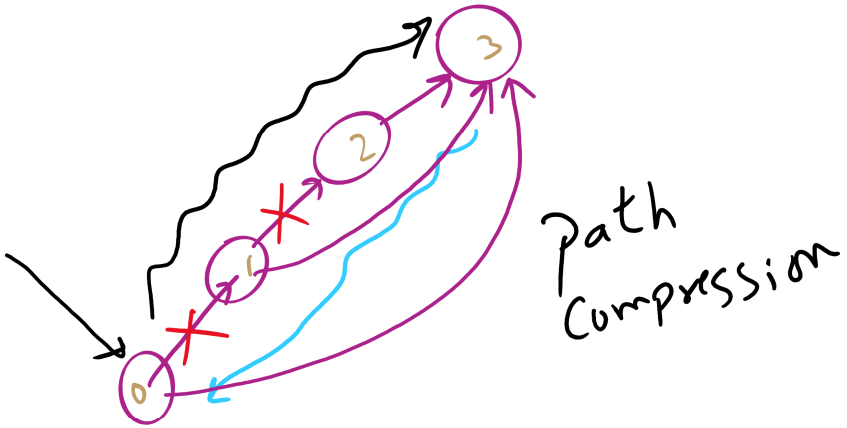
union (i, j)

```
r_i = find(i)
r_j = find(j)
if ( sz[r_i] > sz[r_j])
    id[r_j] = r_i
    sz[r_i] += sz[r_j]
else
    id[r_i] = r_j
    sz[r_j] += sz[r_i]
Count--
```

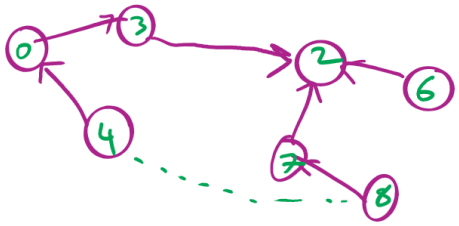
Weighted tree approach analysis

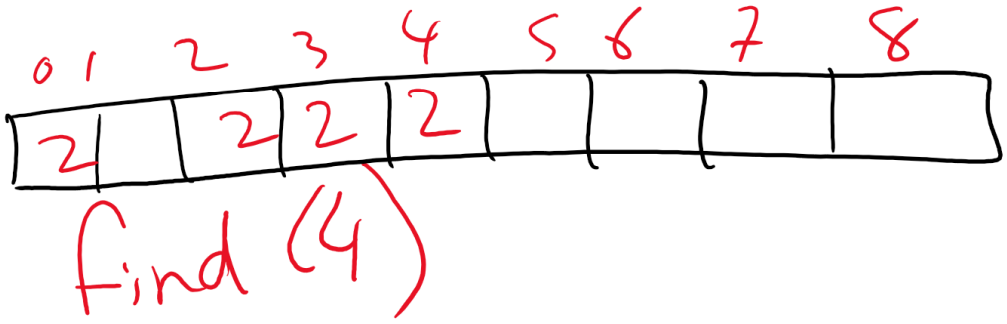
- Runtime?
  - find()?
  - union()?

- Can we do any better?



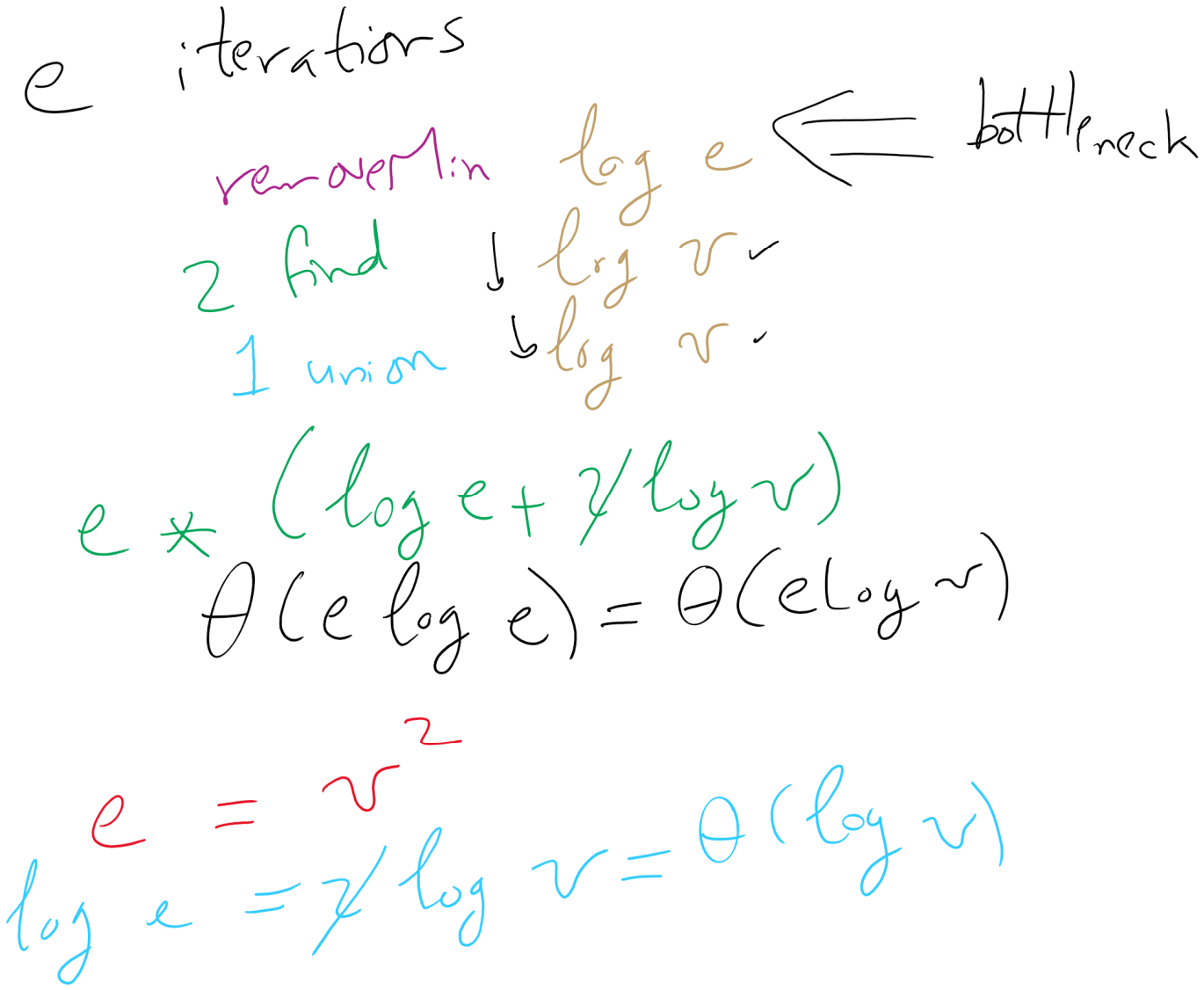
Path Compression Example





Kruskal's algorithm, once again

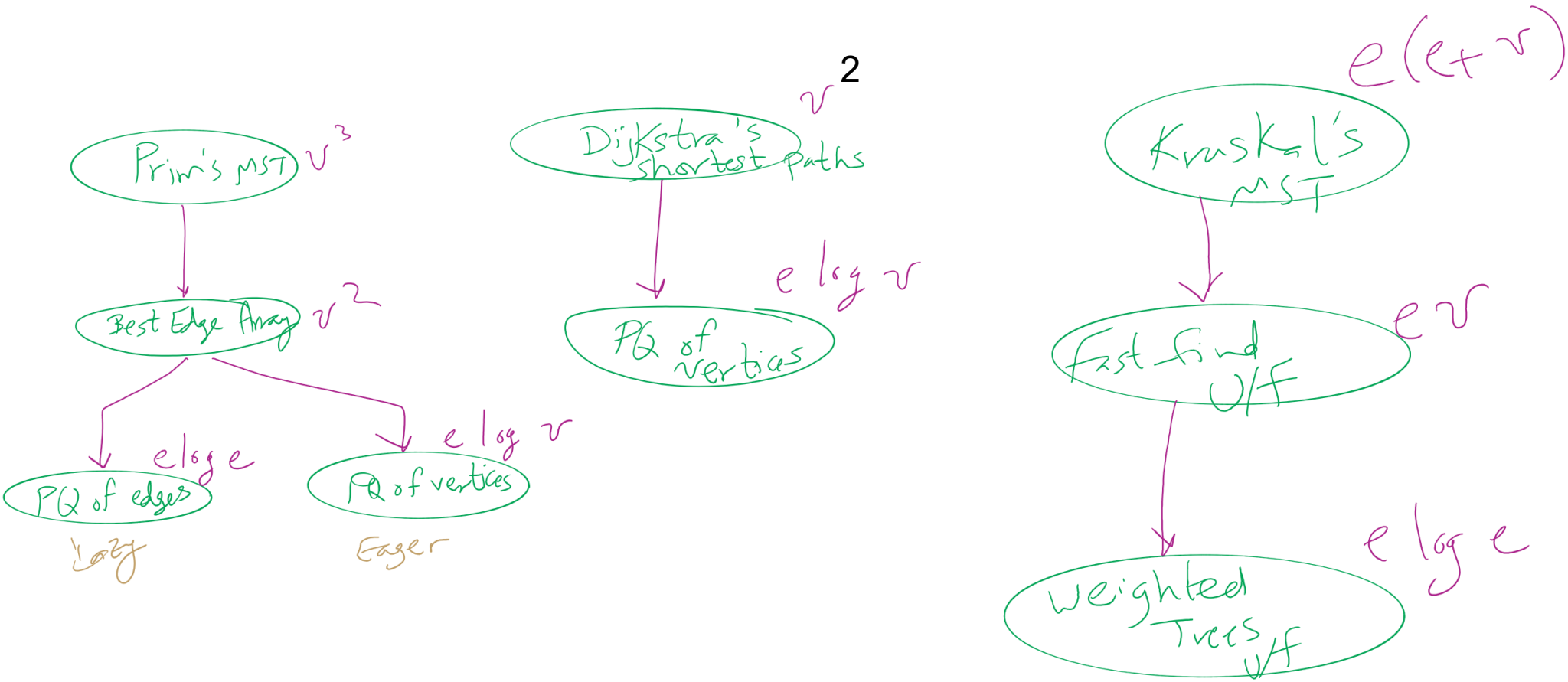
- What is the runtime of Kruskal's algorithm??



Union-Find Implementations

	find	union
Fast-Find	$\Theta(1)$	$\Theta(n)$
Forest of Trees	$\Theta(n)$	$\Theta(n)$
Weighted Trees	$\Theta(\log n)$	$\Theta(\log n)$
Path Compression	$\sim \Theta(1)$	$\sim \Theta(1)$

Algorithm Optimizations



Problem of the Day 1: Weighted Shortest Path

- Input:
  - A road network
    - Road segments and intersections
    - Road segments are labeled by travel time
      - From length and maximum speed
      - How do we get max speed?
  - Starting address and destination address
- Output:
  - A shortest path from source to destination

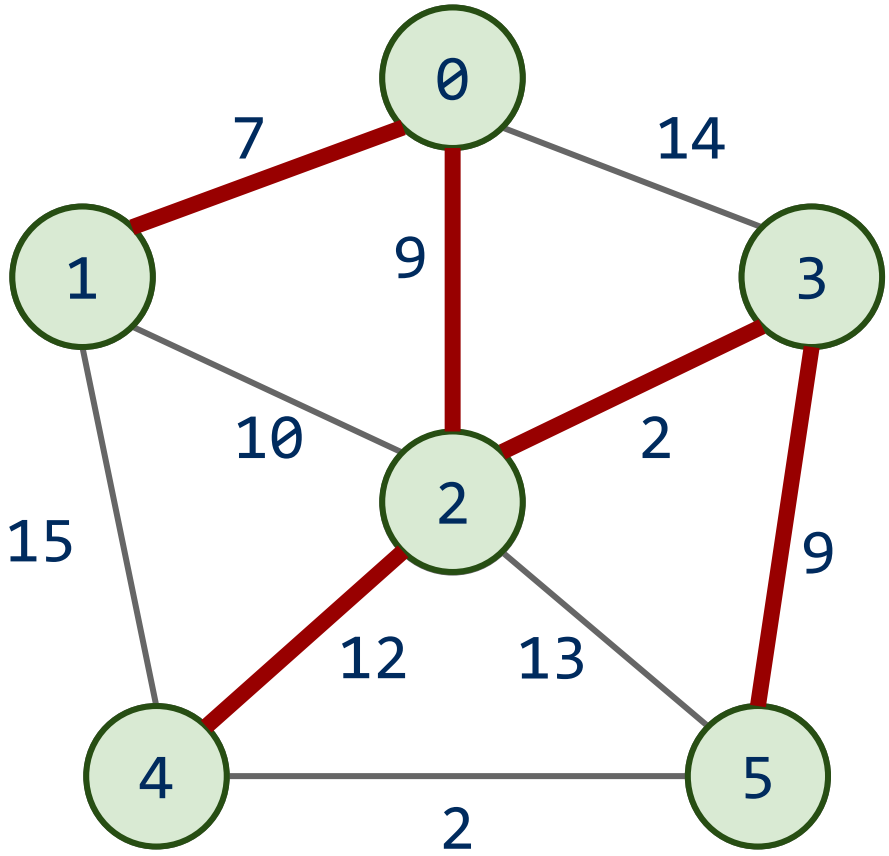
Weighted shortest path

- Dijkstra's algorithm:
  - Set a distance value of MAX_INT for all vertices but start
  - Set cur = start
  - While destination is not visited:
    - For each unvisited neighbor of cur:
      - Compute tentative distance from start to the unvisited neighbor through cur
      - Update any vertices for which a lesser distance is computed
    - Mark cur as visited
    - Let cur be the unvisited vertex with the smallest tentative distance from start

Tentative Distance

distance[cur] + edge weight between cur & neighbor

Dijkstra's example

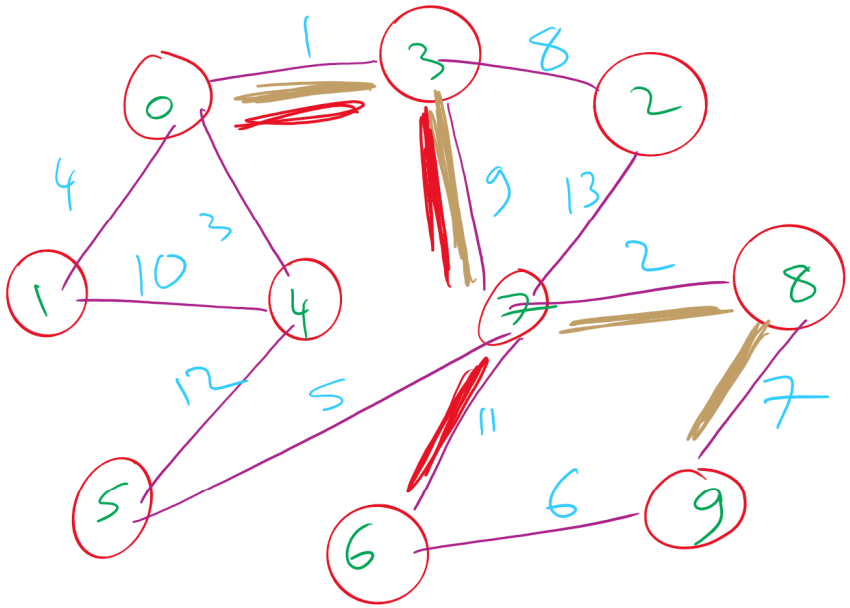


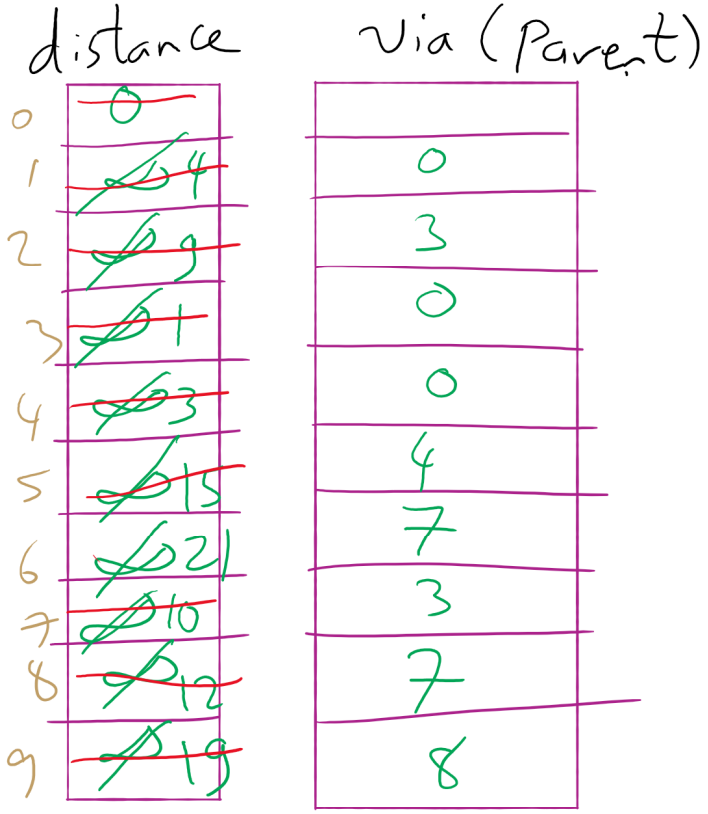
	Distance	Parent
0	0	--
1	7	0
2	9	0
3	11	2
4	21	2
5	20	3

Analysis of Dijkstra's algorithm

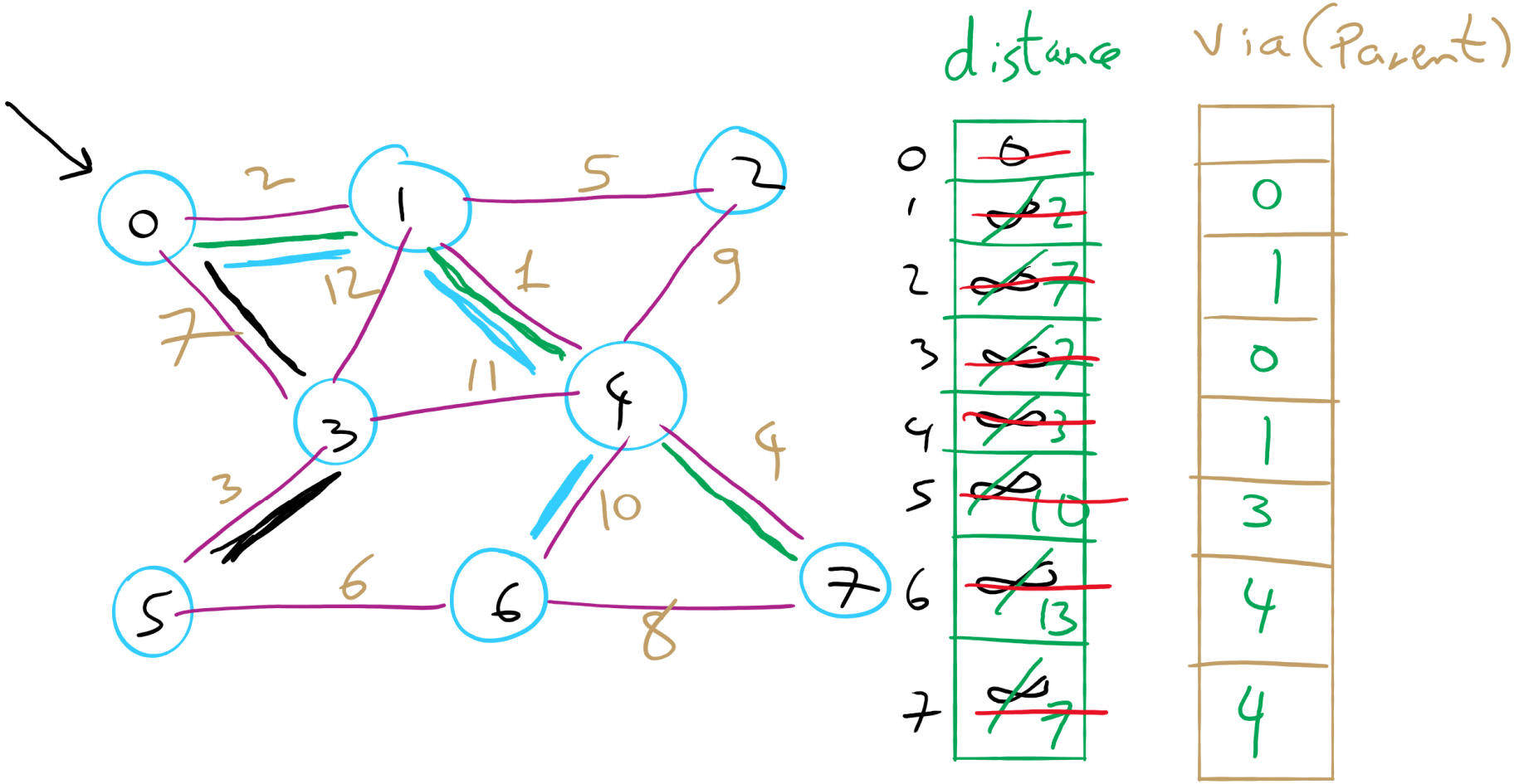
- How to implement?
  - Best path/parent array?
    - Runtime?
  - PQ?
    - Turns out to be very similar to Eager Prims
      - Storing paths instead of edges
    - Runtime?

Dijkstra's Shortest Paths Example 1





Dijkstra's Shortest Paths Example 2



Please submit your reflections by using the CourseMIRROR App

If you are having a problem with CourseMIRROR, please send an email to coursemirror.development@gmail.com

8/29/2022